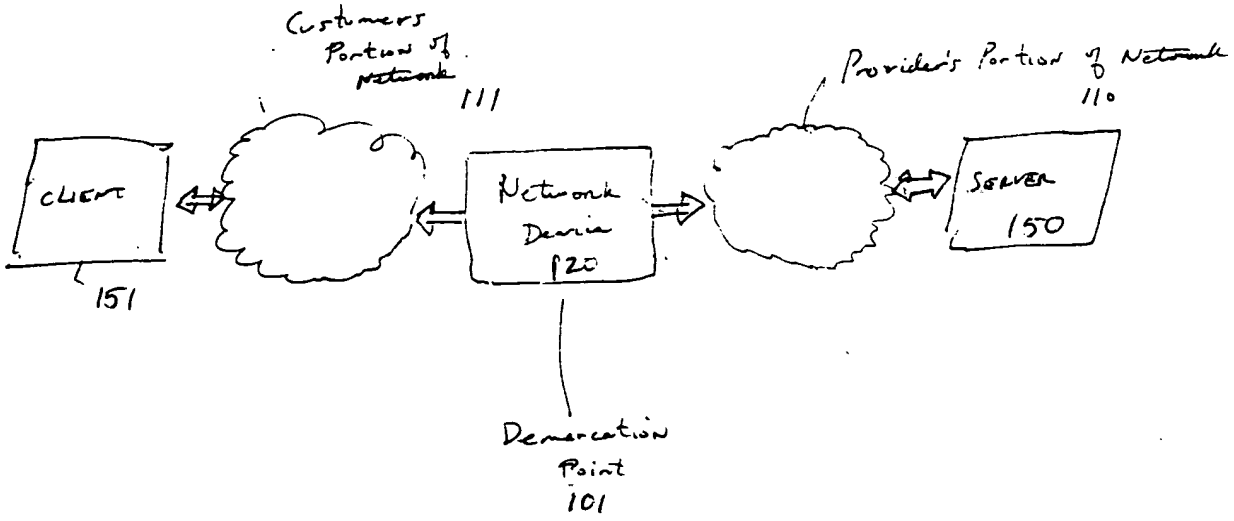

Customers Portion of Network
111
Provider's Portion of Network
110
CLIENT
151
Network Device
120
SERVER
150
Demarcation Point
101

Figure 1

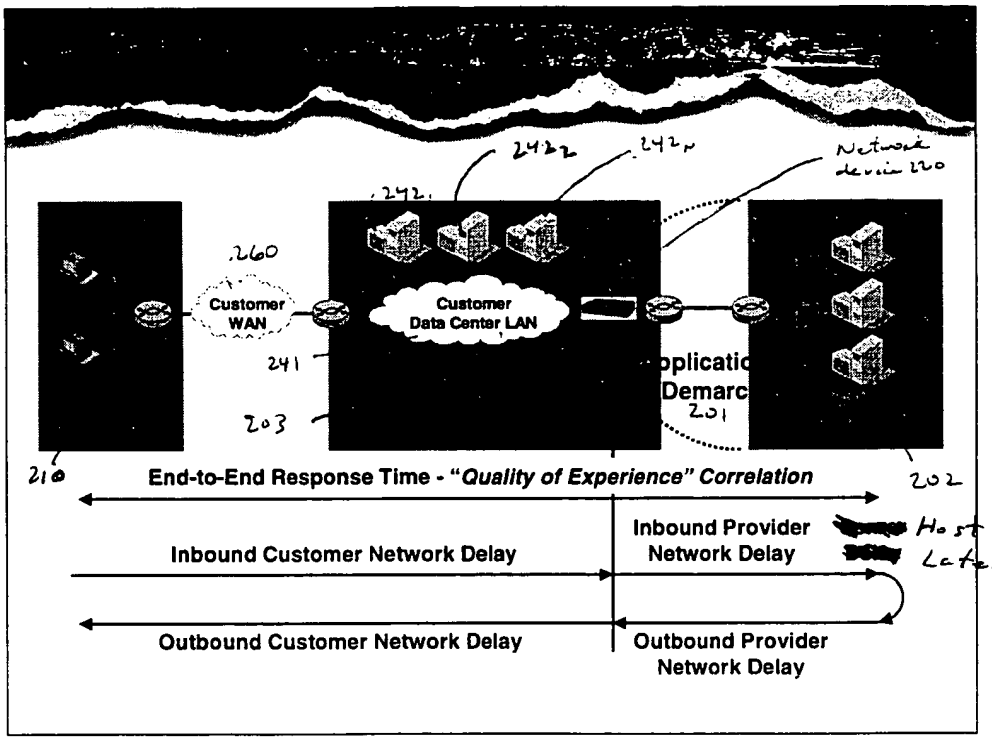
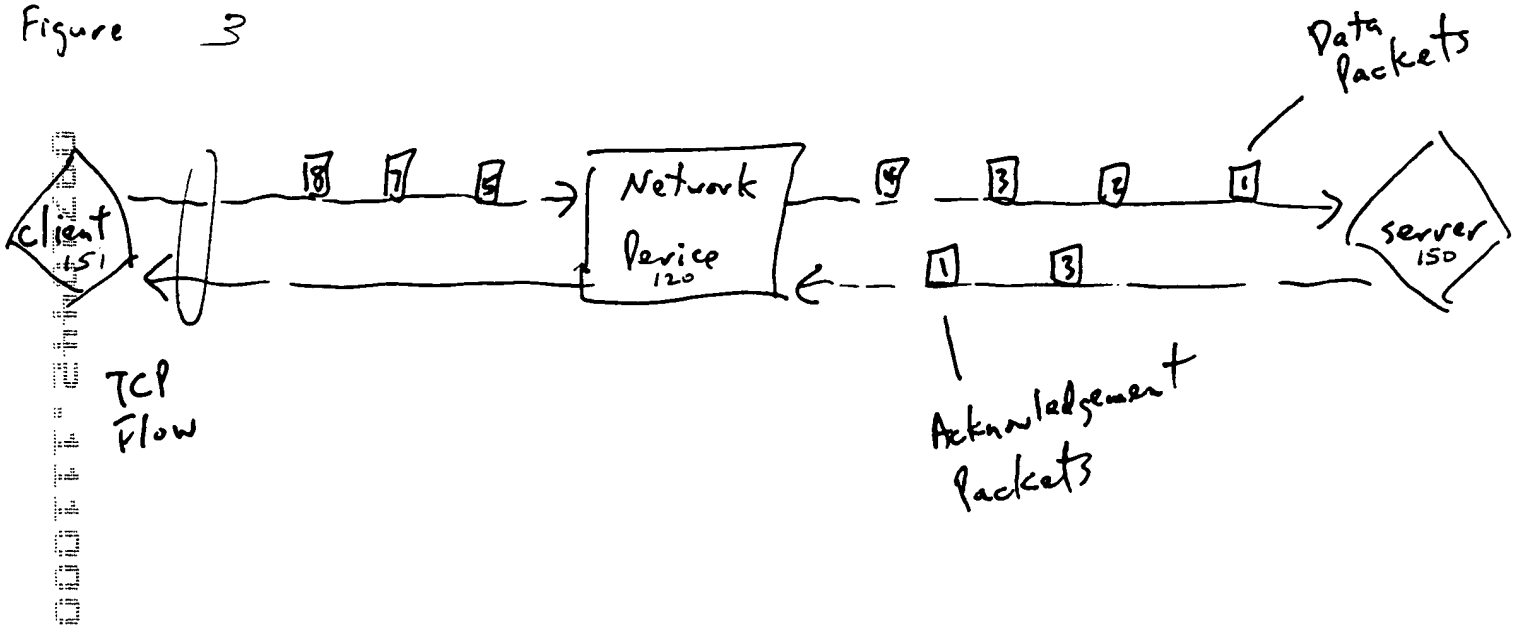

242₂
242ₙ
Network device 220
242₁
260
Customer WAN
Customer Data Center LAN
241
203
Application Demarc
201
210
202
End-to-End Response Time - "Quality of Experience" Correlation
Inbound Customer Network Delay
Inbound Provider Network Delay
Host Latency
Outbound Customer Network Delay
Outbound Provider Network Delay

Figure 2

Figure 3


Data Packets
client 151
Network Device 120
server 150
TCP Flow
Acknowledgement Packets

drixoral.c

```
/*****************************************************************************
 *
 * Module Name: drixoral.c
 *
 * Abstract:    Congestion Index Measurement
 * Author:      Guy Riddle
 * Created:     991102
 *
 * $Revision: 1.1.4.6 $ $Name:  $
 * $Date: 2000/0?/22 22:20:24 $ Copyright Packeteer, Inc. 1999
 *****************************************************************************
 */

#include <artist.h>
#include <mecom.h>
#include <kmalloc.h>
#include "../tcp.h"

#if _PSAPPV

#define MY_DEBUG_LEVEL  gTcpRTMDebugLevel

    extern int          gTcpRTMDebugLevel;

#define meTclassCongestionAccum(tc, msec)       \
            meTclassPktExchMsecsAccum(tc, msec),meTclassPktExchSamplesAccum(tc, 1)


#define DRIX_SEQ_SPOTS      7   /* chosen to avoid kmalloc roundup wasteage */

    typedef struct {
        SEQ             lastseq;
        SEQ             lastack;
        SEQ             seq[DRIX_SEQ_SPOTS];
        TICK            time[DRIX_SEQ_SPOTS];
    } CongestionMeasurement, *CongestionMeasurementPtr;

    void
drixoralMeasureCongestion(
    TCB_INFO_PTR                info
){
    TCB_PTR                     tcb;
    ArtMeasurementPtr           am;
    CongestionMeasurementPtr    drix;
    TCLASS_PTR                  tc;
    DIRECTION                   dir;
    SEQ                         seq;
    UINT32                      msec;
    int                         i, leave;

    whereStr("xxxMeasureCongestion");

    tcb = info->tcb;
    am = &tcb->artData;
    dir = info->dir;

    blurt3("seq %d ack %d flags %X", info->header.seq, info->header.ack, info->header.flags)
;
    blurt4("dir %d len %d end %d tcb %X", dir, info->header.dataLen, info->header.seq + info
->header.dataLen, tcb);

    if(info->header.dataLen > 0){
        if(!am->drixoral[dir])
            am->drixoral[dir] = kmallocz(sizeof(CongestionMeasurement), M_DRIXORAL);

        if((drix = am->drixoral[dir])){
            seq = info->header.seq + info->header.dataLen;
            if(info->header.flags & (TCP_F_SYN|TCP_F_FIN))
                seq++;

            if(SEQ_GT(seq, drix->lastseq) || !drix->lastseq){
                for(i = 0; i < DRIX_SEQ_SPOTS; i++)
                    if(!drix->seq[i])
                        break;

                if(i < DRIX_SEQ_SPOTS){
                    drix->seq[i] = seq;
                    drix->time[i] = info->bcb->tick.ticks;
                }else
                    info2("req %d spot overflow tcb %X", seq, tcb);

                drix->lastseq = seq;
            }
        }
    }

    dir = OTHER_DIRECTION(info->dir);

    if(IS_FLAG_SET(info->header.flags, TCP_F_ACK) && (drix = am->drixoral[dir])
      && (SEQ_GT(info->header.ack, drix->lastack) || !drix->lastack)){
        leave = 0;

        for(i = 0; i < DRIX_SEQ_SPOTS; i++)
            if(!drix->seq[i])
                continue;
            else if(SEQ_EQ(info->header.ack, drix->seq[i])){
                if(info->header.dataLen == 0){
                    msec = TICKS_TO_MSECS_ROUNDED(info->bcb->tick.ticks - drix->time[i]);

                    if((tc = tclassIdToTclassFast(tcb->halfConns[dir].gear.classId))){

                        meTclassCongestionAccum(tc, msec);

                        blurt4("sample %d msec %d class %s dir %d", i, msec, tc->name, dir);
                    }else
                        attn1("no class for TCB %X", tcb);
                }

                drix->seq[i] = 0;
            }else if(SEQ_GT(info->header.ack, drix->seq[i]))
                drix->seq[i] = 0;
            else
                leave++;

        if(leave)
            drix->lastack = info->header.ack;
        else{
            kfree(drix);
            am->drixoral[dir] = 0;
        }
    }
}

    void
drixoralCleanup(
    TCB_PTR                     tcb
){
    ArtMeasurementPtr           am;
    CongestionMeasurementPtr    drix;

    am = &tcb->artData;

    if((drix = am->drixoral[DIR_INBOUND])){
        kfree(drix);
        am->drixoral[DIR_INBOUND] = 0;
    }

    if((drix = am->drixoral[DIR_OUTBOUND])){
        kfree(drix);
        am->drixoral[DIR_OUTBOUND] = 0;
    }
}

#endif /* _PSAPPV */
```

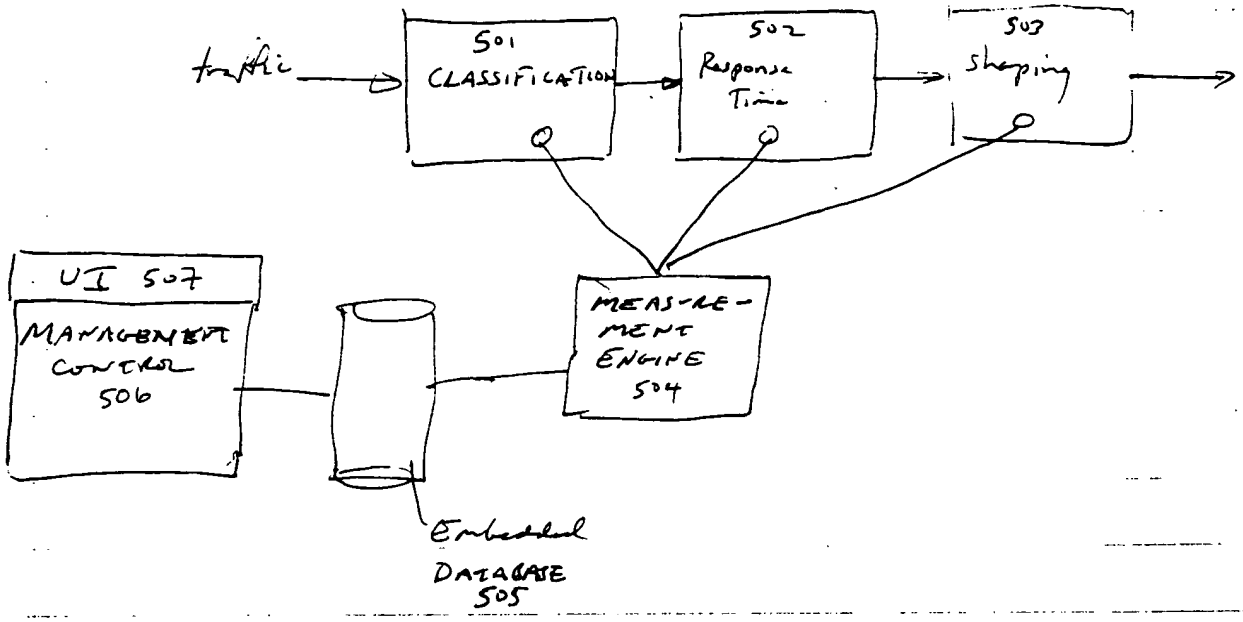

traffic
501
CLASSIFICATION
502
Response Time
503
Shaping
UI 507
MANAGEMENT CONTROL 506
MEASURE-MENT ENGINE 504
Embedded DATABASE 505

Figure 5